[redacted]

Verification of CCSL Specifications

Ling YIN

華東師範大學
EAST CHINA NORMAL UNIVERSITY

Université
Nice
Sophia Antipolis

Objective

❑ Exhaustive Verification with CCSL

- Observer-based (LCTES’09, SIES’10)
  - Verify that a property specified in CCSL holds for a given implementation
- **ICECCS’11**
  - Verify that a property specified in LTL holds for a given CCSL specification => SPIN
  - Means: Transformation into Promela
    - – Pro: Promela supports non-deterministic choice
    - – Pro: Promela is used in TrustableMDA
    - – Con: Promela is asynchronous, does not natively support simultaneity

CCSL -> Promela

- Get some inspiration from the operational semantics of CCSL
  - CCSL clocks: encoded as shared boolean variables
  - A *run* :
    - a sequence of coincident instants
    - valid evolution conforming to the specification
    - Promela must explore ALL the valid runs
  - A *coincident instant*
    - consists of several valid configurations
    - each configuration is a set of ticking decisions, {a,¬b}
    - which configuration is chosen is non-deterministic
  - A *step* :
    - Decide what clocks **MUST** or **CANNOT** fire (enabled)
    - Choose what clocks **ACTUALLY** fire (firing)
      - Non-deterministic choice
      - Conflicts

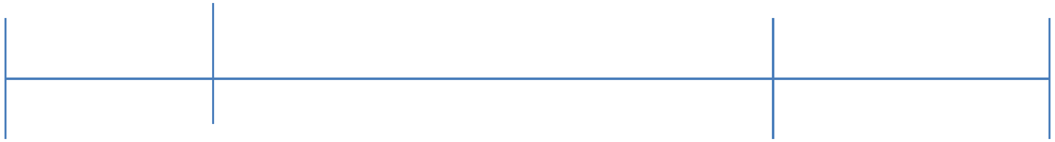
```
typedef Clock { bool must_tick, cannot_tick, actually_tick, dead };
```

CCSL -> Promela

- Global clock declaration

```
typedef Clock { bool must_tick, cannot_tick, actually_tick, dead };
```

- Operator process instantiations + init process
- A *coincident instant*



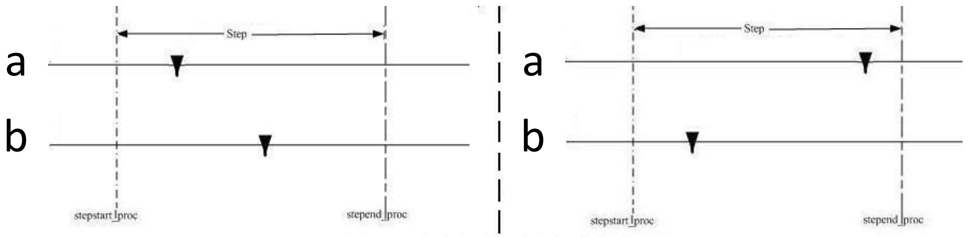
Start *Firing* *End*

Start: compute ticking decisions(must,cannot)

Firing: chose what clocks actually fire, non-determinsitic

End: update+reset

Order:

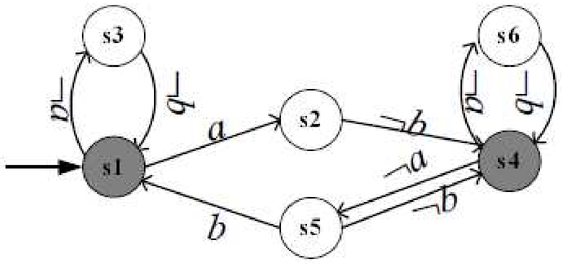


a ticks in coincidence with b in the step

a $\boxed{\sim}$ b

Example

- T={S,A,->,I,clp}, each transition is labeled by a set of actions, representing clock decisions in the coincident instant;clp indicates checkpoints



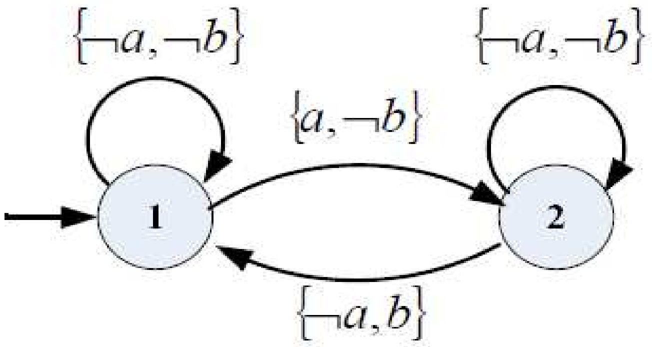
```
proctype alternatesWith(int cLeft; cRight) {
        bool state = true;
        do
        :: start_proc?true;
           if
           :: state -> clocks[cRight].cannot_tick = true ;
           :: ! state -> clocks[cLeft]:cannot tick = true ;
           fi;
           end_proc?true;

           if
           :: state -> if :: clocks[cLeft].actually_tick -> state = false
                       :: else -> skip     fi
           :: !state -> if :: clocks[cRight].actually_tick -> state = true
                       :: else -> skip     fi
           fi
        od
}
```

Enabling

Global non-deterministic choice

State Update

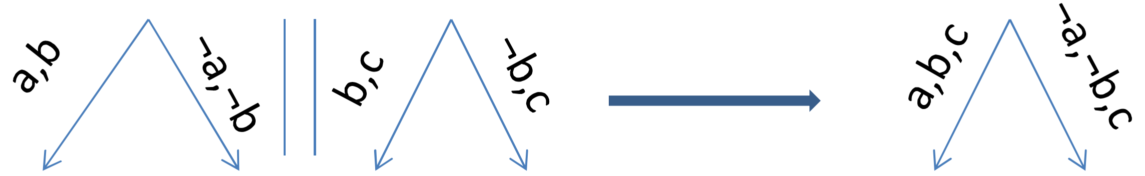


Composition

$T_1 = \{S_1, \mathcal{A}_1, \Rightarrow_1, I_1, C_1\}$ [redacted]

$$T_1 || T_2 = \{S_1 \times S_2, \mathcal{A}_1 \cup \mathcal{A}_2, \Rightarrow, I_1 \times I_2\}$$

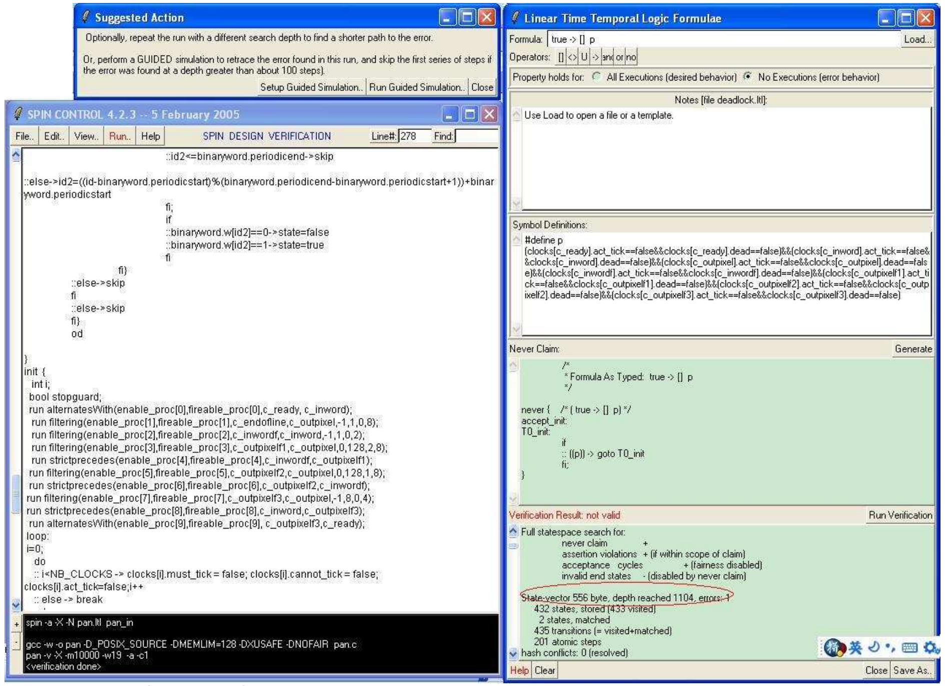
$$\frac{s_1 \xRightarrow{\mu_1} s_1' \in T_1, s_2 \xRightarrow{\mu_2} s_2' \in T_2,\ \forall a \in \mathcal{A}_1 \cup \mathcal{A}_2,\ a \in \mu_1 \wedge\ \neg a \notin \mu_2}{(s_1, s_2) \xRightarrow{\mu_1 \cup \mu_2} (s_1', s_2')}$$



Verifying LTL properties on CCSL

$$\varphi ::= true | inst \wedge \psi | \mathbf{X}\varphi | \mathbf{F}\varphi | \mathbf{G}\varphi | \varphi_1 \mathbf{U} \varphi_2$$
$$\psi ::= c.act\_tick | \neg\psi | \psi_1 \wedge \psi_2$$

- Special variable `inst' guaranteeing properties are checked each `coincident instant'



Optionally, repeat the run with a different search depth to find a shorter path to the error.

Or, perform a GUIDED simulation to retrace the error found in this run, and skip the first series of steps if the error was found at a depth greater than about 100 steps).

```
                                        ::id2<=binaryword.periodicend->skip

::else->id2=((id-binaryword.periodicstart)%(binaryword.periodicend-binaryword.periodicstart+1))+binar
yword.periodicstart
                                        fi;
                                        if
                                        ::binaryword.w[id2]==0->state=false
                                        ::binaryword.w[id2]==1->state=true
                                        fi
                              fi)
                    ::else->skip
                    fi
                    ::else->skip
                    fi)
                    od

}
init {
  int i;
 bool stopguard;
 run alternatesWith(enable_proc[0],fireable_proc[0],c_ready, c_inword);
  run filtering(enable_proc[1],fireable_proc[1],c_endofline,c_outpixel,-1,1,0,8);
  run filtering(enable_proc[2],fireable_proc[2],c_inwordf,c_inword,-1,1,0,2);
  run filtering(enable_proc[3],fireable_proc[3],c_outpixelf1,c_outpixel,0,128,2,8);
  run strictprecedes(enable_proc[4],fireable_proc[4],c_inwordf,c_outpixelf1);
 run filtering(enable_proc[5],fireable_proc[5],c_outpixelf2,c_outpixel,0,128,1,8);
 run strictprecedes(enable_proc[6],fireable_proc[6],c_outpixelf2,c_inwordf);
run filtering(enable_proc[7],fireable_proc[7],c_outpixelf3,c_outpixel,-1,8,0,4);
run strictprecedes(enable_proc[8],fireable_proc[8],c_inword,c_outpixelf3);
 run alternatesWith(enable_proc[9],fireable_proc[9], c_outpixelf3,c_ready);
loop:
i=0;
  do
  :: i<NB_CLOCKS -> clocks[i].must_tick = false; clocks[i].cannot_tick = false;
clocks[i].act_tick=false;i++
  :: else -> break
```

```
spin -a -X -N pan.ltl  pan_in

gcc -w -o pan -D_POSIX_SOURCE -DMEMLIM=128 -DXUSAFE -DNOFAIR  pan.c
pan -v -X -m10000 -w19  -a -c1
<verification done>
```

Formula: true -> [] p

Property holds for: ○ All Executions (desired behavior) ● No Executions (error behavior)

Notes [file deadlock.ltl]:

Use Load to open a file or a template.

Symbol Definitions:

```
#define p
(clocks[c_ready].act_tick==false&&clocks[c_ready].dead==false)&&(clocks[c_inword].act_tick==false&
&clocks[c_inword].dead==false)&&(clocks[c_outpixel].act_tick==false&&clocks[c_outpixel].dead==fals
e)&&(clocks[c_inwordf].act_tick==false&&clocks[c_inwordf].dead==false)&&(clocks[c_outpixelf1].act_ti
ck==false&&clocks[c_outpixelf1].dead==false)&&(clocks[c_outpixelf2].act_tick==false&&clocks[c_outp
ixelf2].dead==false)&&(clocks[c_outpixelf3].act_tick==false&&clocks[c_outpixelf3].dead==false)
```

Never Claim:

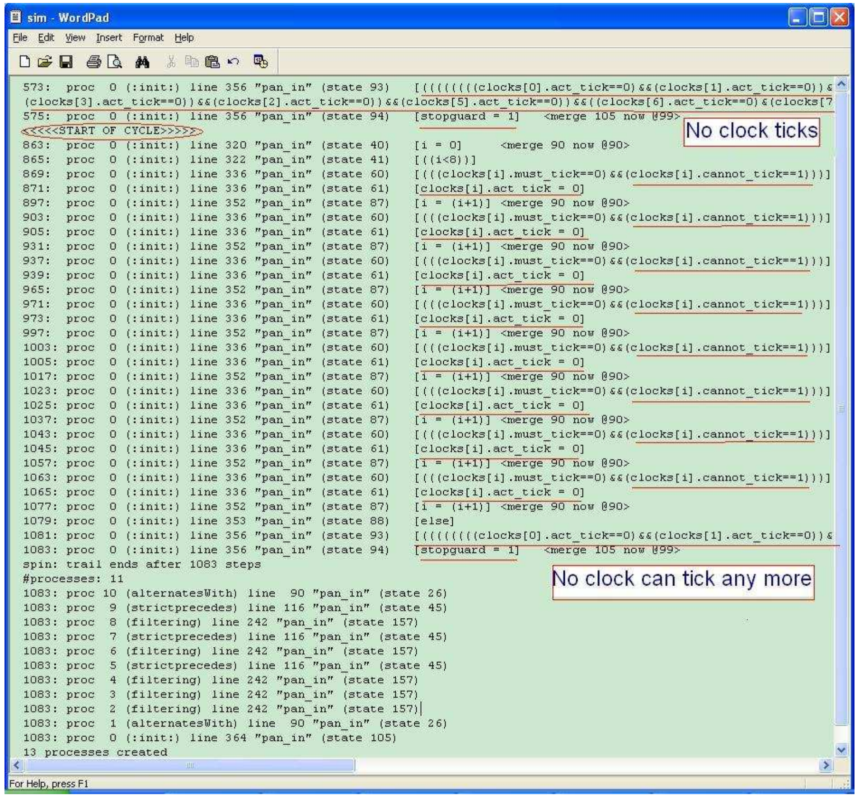
```
            /*
             * Formula As Typed:  true -> [] p
             */

never {    /* ( true -> [] p) */
accept_init:
T0_init:
            if
            :: ((p)) -> goto T0_init
            fi;
}
```

Verification Result: not valid

```
Full statespace search for:
            never claim             +
            assertion violations    + (if within scope of claim)
            acceptance   cycles                + (fairness disabled)
            invalid end states    - (disabled by never claim)

State-vector 556 byte, depth reached 1104, errors: 1
    432 states, stored (433 visited)
      2 states, matched
    435 transitions (= visited+matched)
    201 atomic steps
hash conflicts: 0 (resolved)
```



No clock ticks

```
573:  proc  0 (:init:) line 356 "pan_in" (state 93)   [((((((((clocks[0].act_tick==0)&&(clocks[1].act_tick==0))&
(clocks[3].act_tick==0))&&(clocks[2].act_tick==0))&&(clocks[5].act_tick==0))&&((clocks[6].act_tick==0)&(clocks[7
575:  proc  0 (:init:) line 356 "pan_in" (state 94)   [stopguard = 1]    <merge 105 now @99>
<<<<<START OF CYCLE>>>>>
863:  proc  0 (:init:) line 320 "pan_in" (state 40)   [i = 0]      <merge 90 now @90>
865:  proc  0 (:init:) line 322 "pan_in" (state 41)   [((i<8))]
869:  proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
871:  proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
897:  proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
903:  proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
905:  proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
931:  proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
937:  proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
939:  proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
965:  proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
971:  proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
973:  proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
997:  proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
1003: proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
1005: proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
1017: proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
1023: proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
1025: proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
1037: proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
1043: proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
1045: proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
1057: proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
1063: proc  0 (:init:) line 336 "pan_in" (state 60)   [(((clocks[i].must_tick==0)&&(clocks[i].cannot_tick==1)))]
1065: proc  0 (:init:) line 336 "pan_in" (state 61)   [clocks[i].act_tick = 0]
1077: proc  0 (:init:) line 352 "pan_in" (state 87)   [i = (i+1)]  <merge 90 now @90>
1079: proc  0 (:init:) line 353 "pan_in" (state 88)   [else]
1081: proc  0 (:init:) line 356 "pan_in" (state 93)   [((((((((clocks[0].act_tick==0)&&(clocks[1].act_tick==0))&
1083: proc  0 (:init:) line 356 "pan_in" (state 94)   [stopguard = 1]    <merge 105 now @99>
spin: trail ends after 1083 steps
#processes: 11
1083: proc 10 (alternatesWith) line  90 "pan_in" (state 26)
1083: proc  9 (strictprecedes) line 116 "pan_in" (state 45)
1083: proc  8 (filtering) line 242 "pan_in" (state 157)
1083: proc  7 (strictprecedes) line 116 "pan_in" (state 45)
1083: proc  6 (filtering) line 242 "pan_in" (state 157)
1083: proc  5 (strictprecedes) line 116 "pan_in" (state 45)
1083: proc  4 (filtering) line 242 "pan_in" (state 157)
1083: proc  3 (filtering) line 242 "pan_in" (state 157)
1083: proc  2 (filtering) line 242 "pan_in" (state 157)
1083: proc  1 (alternatesWith) line  90 "pan_in" (state 26)
1083: proc  0 (:init:) line 364 "pan_in" (state 105)
13 processes created
```

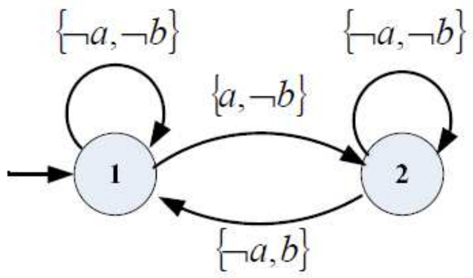
No clock can tick any more

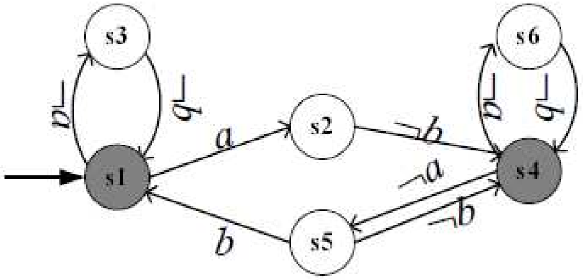
For Help, press F1

Encoding correctness

LTL property pattern

+ coincident encoding





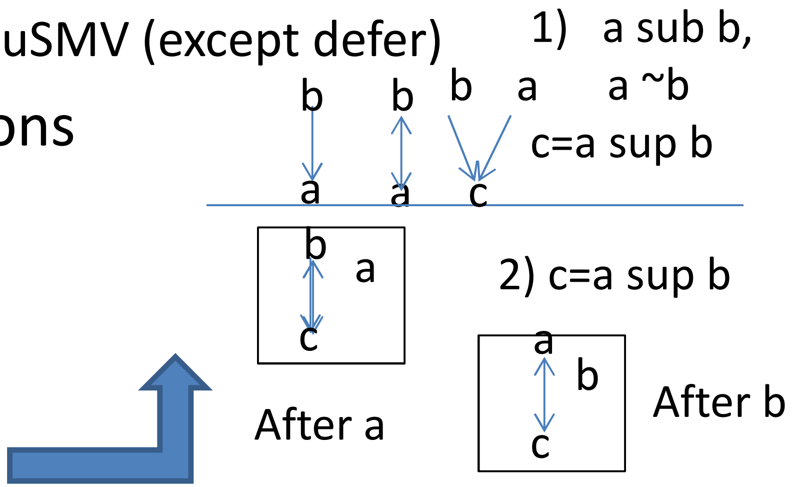
Checkpoint bisimulation equivalent

- Checkpoint bisimulation checks from checkpoint to checkpoint, requiring compared systems have executed the same set of visible actions. Orders of the actions are irrelevant
- It preserves logical truth under the pattern
- It is a congruence w.r.t parallel composition

Discussion

❑ Synchronous Transition System

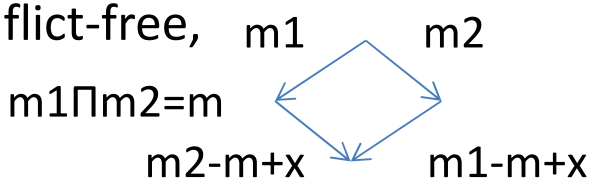
- easier with synchronous models, NuSMV (except defer)



❑ Choices among valid configurations

Unpredictable random -> predictable

- Conflict-free,

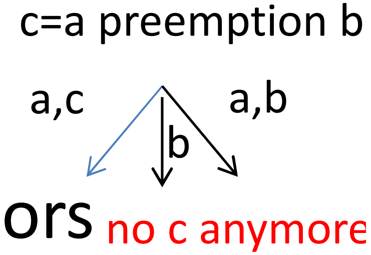


Condition: for all states, each pair of transitions, (m1-m,m2-m) independent

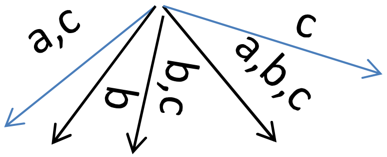
Independent: 1)Not connected, don’t affect common clocks (too strong, e.g. prevent c=a sup b) → 2) build dependent relations for each state, only one instant (still strong, prevent c=a union b, may cause problem on strictSampling)

- Otherwise,
  - Some clocks tick in some paths, while can not tick in others (deadlock or not)
  - If then else case

Non Conflict-free examples



- Single operator: sampling, preemption
- Conflict caused by non conflict-free operators
  - a sub c, d=c filterby(01)^w, d=f preemption b



b blocks d->c->a

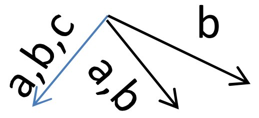
ac,b involved in disjoint operators.

(b in operator o3, while a and c involved in operator o1 and o2.
)

Non Conflict-free examples

- Composition of conflict-free operators:
  - 1) Choosing one path blocks unchosen clocks

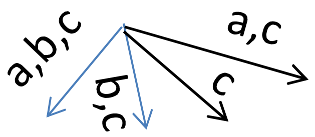
b= a wait 1, c sub b



no c anymore

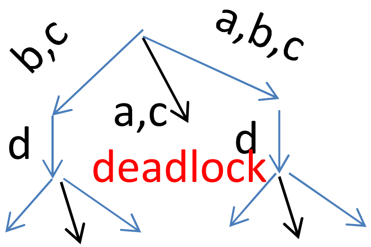
d=c filterby (01)^w,

a sub c, b sub c, b<d

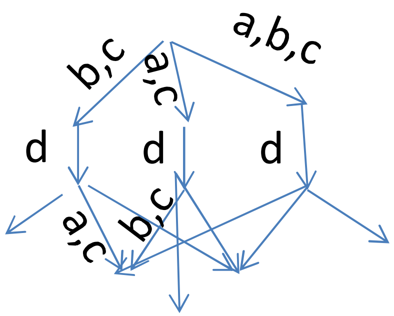


block d-> c->a,b

c=a union b, c ~ d, b <d



deadlock

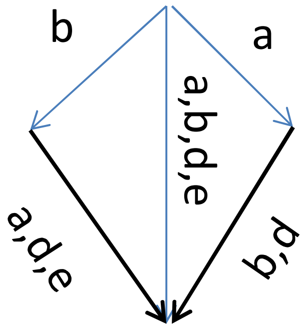


But removing d<d (lower one) yields conflict-free, even it does not satisfy the definition used above. Right now, ignore this case in the definition.

Non Conflict-free examples

- 2)Choosing one path doesn't block unchosen clocks, but forcing different new clocks

  d=a sup b, e=a intersection d



- If then else case:

  a sub m, b sub n

  c=a union b, b<d, a<d, c ~d, a#b

Looking for condition for conflict-free

- Not composition preserving
  - Composition of self conflict-free operators may introduce conflict
  - Restrict conflict operators may end as non conflict-free specification
    - c=a strictSampling b, a~b